Un algorithme primitif récursif pour la fonction Inf

René David

Un algorithme primitif récursif pour la fonction Inf

R David (*)
Laboratoire de Mathématiques
Université de Chambéry
73376 Le Bourget du Lac Cedex
email : david@univ-savoie.fr

Abstract : We give a primitive recursive algorithm (for the structure of integers - in unary - and lists of integers) that computes the minimum of two integers in linear time of this minimum. This algorithm refutes a conjecture of L.Colson.

Résumé : Nous donnons un algorithme primitif récursif (pour la structure des entiers - en unaire - et des listes d'entiers) qui calcule le minimum de deux entiers en temps linéaire en le minimum de ces deux entiers. Cet algorithme réfute une conjecture de L.Colson.

Introduction

La plupart des systèmes de programmation sont équivalents au regard de la classe des fonctions qu'ils permettent de calculer, que ce soit des modèles "théoriques" (machines de Turing, lambda calcul, ...) ou "réels" (langages de programmation impératifs (Pascal, C, ...) ou fonctionnels (Lisp, ML, ...).
Certaines restrictions limitent - en théorie - cette puissance de calcul: les fonctions primitives récursives forment un sous ensemble strict de l'ensemble des fonctions récursives (c'est même un sous ensemble strict de l'ensemble des fonctions récursives totales); le lambda calcul typé (par exemple avec le système F de typage du second ordre) ne permet que de calculer les fonctions récursives dont la totalité est prouvable dans l'arithmétique de Peano du second ordre. Dans la pratique cette restriction n'est pas génante puisque toutes les fonctions récursives dont le temps calcul (sur une machine de Turing) est majoré par une fonction "raisonnable" (une fonction élémentaire est en ce sens raisonnable !) en la taille des données font partie de ces classes.
Il y a cependant une différence essentielle entre le fait de pouvoir - dans un système donné - calculer une fonction et y représenter un algorithme particulier.

(*) Ce travail a été partiellement supporté par l'URA de logique de Paris 7 et le LIP de l'ENS Lyon

L.Colson (voir [1,2]) a ainsi montré que, bien que la fonction Inf qui calcule le minimum de 2 entiers soit évidemment primitive récursive, il n'est pas possible de représenter - dans le cadre des "algorithmes" primitifs récursifs- le bon algorithme, i.e celui qui décrémente alternativement chacun des 2 arguments . Son théorème " d'ultime obstination" montre en effet que tout algorithme primitif récursif doit nécessairement "choisir" un des arguments (pour un sens précis de cette notion voir [1,2]), ce qui a pour conséquence immédiate qu'aucun algorithme primitif récursif ne peut calculer la fonction Inf en un temps qui n'est fonction que du minimum et donc en le " bon " temps.
Que devient ce problème quand on étend le pouvoir d'expression des algorithmes ? L.Colson a montré que si on autorise (dans la récurrence primitive) des paramètres fonctionnels (c'est à dire si le système est le premier niveau du systeme T de Gödel) on peut - aisémment - représenter le bon algorithme pour calculer la fonction Inf.
Il donne également un algorithme qui calcule cette fonction en temps $O((inf)^2)$ si on autorise un type de données supplémentaire: les listes d'entiers et conjecture que dans ce cadre le carré du minimum est nécessaire. La motivation en est qu'il semble que tout algorithme ne puisse " garder en mémoire " que les calculs déja faits sur au plus un argument; un tel algorithme devra donc nécessairement " reprendre à zéro " des calculs déja faits; le temps calcul sera alors au moins la somme des n premiers entiers - si n est le minimum - et donc le carré de ce minimum. Bien que cette propriété soit vraie (pour une définition précise de cette propriété et une preuve voir [3]) cet argument ne suffit en fait pas à prouver la conjecture.
Nous donnons en effet ci-dessous un algorithme primitif récursif (sur les entiers en unaire avec listes) qui calcule la fonction Inf en temps O(inf).
Nous ne donnerons pas ici (voir [1,2]) la définition précise de la notion d "algorithme" primitif récursif et de son temps de calcul. On se contentera de dire que cela consiste à voir une définition primitive récursive de la fonction (à partir des fonctions de base (qui sont ici les fonctions constantes 0 - pour le type entier- et nil - pour le type liste-, les fonctions successeur et cons (de type : entier, liste->liste) et les fonctions de projection) , de la composition et de la récurrence primitive c'est à dire : f(o,x)=g(x) et f(n+1,x)=h(n,x,f(n,x)) pour les entiers et f(nil,x)=g(x) et f(cons(n,l),x)=h(n,l,x,f(l,x)) pour les listes où x est une liste de paramétres de types quelconques) comme un système de réécriture où l'on oriente les égalités ci-dessus de la gauche vers la droite; la stratégie de réduction étant celle de l'appel par nom. La complexité de l'algorithme mesure le nombre de fois où les règles de réécriture sont utilisées .

Des calculs sur ordinateurs indiquent que la constante (représentée par le O de O(inf)) est de l'ordre de 30 .
Il est important de noter ici que l'algorithme décrit ci-dessous ne calcule le minimum en temps O(inf) que si la stratégie utilisée est celle de l'appel par nom .

Idée de l'algorithme

1) utiliser la liste [1,2,3,...,n] pour comparer n et m avec les puissances successives de 2 . On utilise, pour cette comparaison, l'algorithme usuel de calcul du Inf (il est appellé " Min " ci-dessous) . Min décrémente son deuxieme argument jusqu'à 0 et fonctionne donc en temps O($2^{\text{ieme argument}}$).
Compte tenu du mode d'évaluation, la liste [1,2,3,...,n] ne sera pas " construite " complétement ; ceci explique le " ne servira jamais " indiqué plus loin .

2) si on trouve un (plus petit) k tel que $n \leqslant 2^k$ et $m > 2^k$ alors on sait que n est plus petit que m . De même si on trouve un (plus petit) k tel que $m \leqslant 2^k$ et $n > 2^k$ alors on sait que m est plus petit que n .
3) si on trouve un k tel que $2^k \leqslant n,m < 2^{(k+1)}$ alors l'algorithme Min permet de conclure .

Le terme complet

-prédecesseur: N->N
pred(0)=0
pred(n+1)=n

-différence : N,N->N
dif(0,n)=n
dif(m+1,n)=pred(dif(m,n))

-test à zero : N,N,N->N
test(0,p,q)=p
test(n+1,p,q)=q

-min : N, N-> N
min(n,m)= test(dif(m,n),n,m)

-double : N->N
double(0)=0
double(n+1)=SS double(n)

-exponentielle : N->N
exp(0)=1
exp(n+1)=double(exp(n))

-plus un : L->L
plus(nil)=nil
plus(cons(a,l))=cons(Sa,plus(l))

-liste de 0 à n : N->L
list(0)=cons(0,nil)
list(n+1)=cons(0,plus(list(n)))

-si n<=2^a alors p sinon q : N,N,N->N
si(a,n,p,q)=test(dif(exp(a),n),p,q)

-itération : L,N,N,N,N->N
iter(nil,n,m,p,q)=0 {ne servira jamais!}
iter(cons(a,l),n,m,p,q)=
si(a,n,
si(a,m,min(n,m),p),
si(a,m,q,iter(l,n,m,p,q)))

-inf: N,N->N
inf(n,m)=iter(list(n),n,m,n,m)

Proposition

L'algorithme décrit ci-dessus calcule la fonction inf en temps O(inf).

Preuve:

Cela résulte facilement des faits suivants

Fait 1 : pour tout n et m il existe k< inf(m,n) tel que
soit (1) $m \leqslant 2^k < n$, soit (2) $n \leqslant 2^k < m$, soit (3) $2^k < n,m \leqslant 2^{(k+1)}$

Fait 2 : exp calcule l'exponentielle de k en temps $O(2^k)$

(preuve immédiate par récurrence sur k)

Fait 3 : si on a la situation (1) ou (2) du fait 1 le temps calcul est $\sum_{i=1}^{k} O(2^i)$

$= O(2^k) = O(inf)$.

si on a (3) le temps calcul est ($\sum_{i=1}^{k+1} O(2^i)$) + O(max(n,m)) = O(inf).

La fonction Inf dans le système F

L.Colson formule dans [1] une autre question : Dans le lambda calcul typé (système F) existe t- il un algorithme calculant la fonction Inf en temps inférieur au carré du minimum ? Nous montrons (voir [4]) qu'on peut trouver un tel algorithme qui fonctionne en temps O (inf.log(inf)). L'idée de base est la même, une difficulté - essentielle - supplémentaire dans ce cadre est que (contrairement au cadre primitif récursif) il n'y a pas de prédecesseur en temps constant sur les entiers de Church du lambda calcul.

Références

[1] L.Colson : Représentation intentionnelle d'algorithmes dans les systèmes fonctionnels. Thèse (université de Paris VII (1991))

[2] " : About intensional behaviour of primitive recursive algorithms LNCS 372 (1989)

[3] R.David : About the asymptotic behaviour of primitive recursive algorithms (en préparation)

[4] " : The Inf function in the system F (manuscript)